(4)

The Design and Implementation of a Special Purpose Cache for the Roll Back Chip

Narayana S. Mani Surya Mantha

17 October 1988

Abstract

The Roll-back chip is an integral part of a special purpose simulation engine based on the Time-Warp mechanism. The RBC is in the critical path of the system and the roll-back processor should not have to access the memory in performing critical operations. We describe the design and implementation (in VLSI) of a special purpose cache that stores the most recently accessed data and performs certain operations on it.

Contents

Chapter 1

Introduction

1.1 The Problem

Computer Simulation of large, complex systems remains a major stumbling block in many research and development efforts today. The need for high performance, coupled with the parallelism inherent in many of the systems being modeled, has made the execution of simulation programs on a parallel processor an area of considerable interest.

1.2 What is Distributed Simulation?

Distributed Simulation is the execution of discrete simulation programs on a parallel processor. This requires partitioning the program into distinct computational units.

The physical system can be visualized as some number of independent, concurrently executing entities, that is, physical processes that interact in some fashion. Each physical process is modeled by a separate simulation program called a logical process. Interactions between physical processes are modeled by timestamped messages exchanged between the corresponding logical processes. The timestamp denotes the point in simulated time when the event occurs in the receiving process. State transitions that only affect the internal behavior of a physical process can be modeled by the process sending a message to itself, allowing all events to be modeled as messages.

This simple mapping demonstrates that **spatial** aspects of the physical

system correspond naturally to a parallel simulation program. It is much more difficult to map temporal aspects however.

1.3 The Time-Warp Mechanism and the Roll Back Chip

In uniprocessor simulation programs, time in the physical system is modeled by a single global variable holding the current value of simulated time. Causality is preserved through strict adherence to the rule that events are processed in nondecreasing timestamp order. This imposes a strict order on the execution of the program that must be relaxed if parallel execution is to be exploited.

The following two problems related to simulated time must be addressed by any distributed simulation program.

1. How can the single global clock variable of uniprocessor simulation programs be replaced by a distributed clock?

2. How does one ensure that the partial ordering of events imposed by causality in the physical system are not violated by the distributed simulator?

The first problem can be solved in two ways. In the first method, a global clock process is used to ensure that all logical processes advance together in lock step. The global clock process repeatedly waits until all activity has ceased in the current timestep and then broadcasts a message allowing processes to advance to the next timestep.

There are a lot of conservative approaches proposed in the literature in which processes cautiously advance their clocks only when they are absolutely certain that this will not violate any causality constraint. We propose the use of the Time Warp scheme that takes a more liberal viewpoint and allows processes to progress as rapidly as possible, sometimes resulting in an incorrect state. On the occurrence of such an error, the computation rolls back in simulated time to a point before the error.

Let us consider an example to illustrate the above. Suppose, that a secretary takes orders from her three bosses A B and C. She gets messages from A and B that she perform certain tasks for them at time t_a and t_b

respectively, which she carries out one after the other immediately. She assumes that she is not going to get a message from C before t_a or t_b. But if some time later, she gets a message from C to perform a task t_c which is before t_a or t_b, she is in trouble since this task could have affected the tasks she did. Thus this task is in conflict with the earlier tasks. She has to undo all the tasks and redo them according to the new specifications. Due to this problem she needed to save a file for each task performed so that when she got a message like the one from boss C, she would just open up the old files she saved and redo the tasks.

In this example, the last order corresponds to the arrival of a message at a process with a timestamp less than that of the last message processed by the process. The undoing of the tasks corresponds to the roll back in the computation that we discussed above.

Rollbacks per se do not represent a serious liability because, a rolled back computation represents time a conservative algorithm would spend blocking the process. However, the state of each process must occasionally be saved regardless of whether or not rollbacks actually occur. State saving overheads are serious and will cripple simulation programs containing large amounts of state. For example, more than a megabyte is required to hold terrain information in large simulations of battlefield scenarios.

No viable software based approach exists to deal with the state saving problem. Current implementations of Time Warp save the entire state of each process after each event. This is clearly infeasible for programs containing a lot of state. Infrequent state saving can reduce this overhead somewhat, but at the cost of severely reducing the efficiency of the rollback mechanism.

The use of special purpose hardware to attack this problem has been proposed by Fujimoto and Gopalakrishnan[?]. A component called the roll back chip(RBC) minimizes state saving and state management overheads in Time Warp. The RBC is a key component of a special purpose, distributed simulation engine (USE: Utah Simulation Engine) that is currently being investigated. The USE uses a message based multicomputer architecture that combines conventional microprocessors, memory and communication coprocessors with application specific IC's.

The Roll-back chip is an integral part of a special purpose simulation engine based on the Time-Warp mechanism. The RBC is in the critical path of the system and the roll-back processor should not have to access the memory in performing critical operations. We describe the design and

implementation (in VLSI) of a special purpose cache that stores the most recently accessed data and performs certain operations on it.

Chapter 2

Functional Description of the Cache

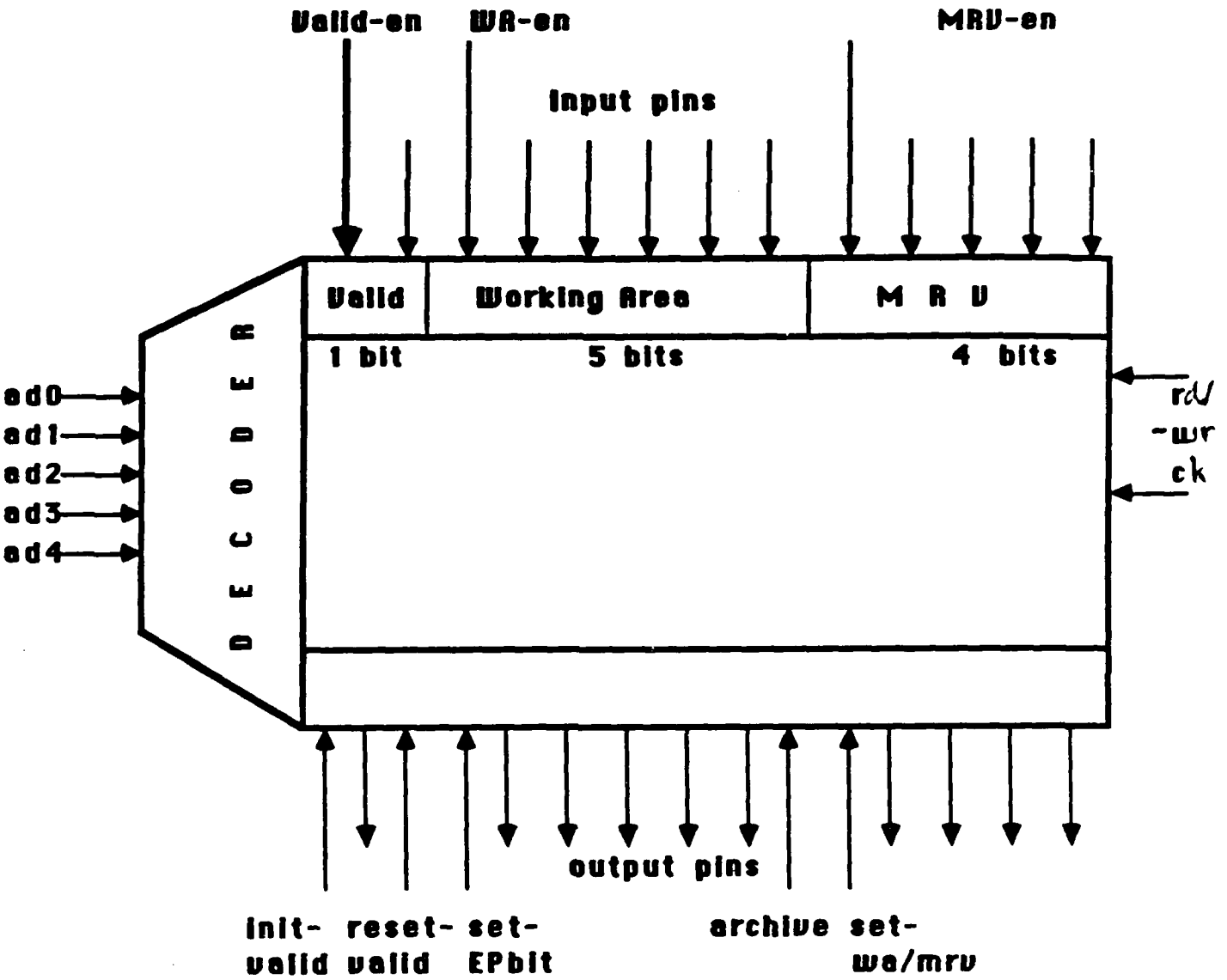
2.1 Internal Structure

The Cache has a decoder that takes an input address and enables one of the 32 entries or rows of the Cache. Each entry or row consists of three main fields(see Fig 2.1).

2.1.1 Fields in a Row of the Cache

A single row in the Cache has the following fields:

- **A:** The **valid** bit is set if that row of the Cache does not hold valid data.
- **WA:** This is a five bit field which gives the current **Working Area** for line specified by the line field. The least significant four bits are used to represent the Working Area number and the most significant bit is an extra precision bit which is used for modulo comparison. This MSB bit will be referred to as the EP bit.
- **MRV:** This four bit field gives the frame number of the **Most Recent Version** of the line.



2.1: BLOCK DIAGRAM OF ROLL BACK CACHE

2.2 Operations on the Cache

The Cache operations are:

- **READ:** The address input to the decoder selects a particular row. The read line is set and the field enable selects a particular field. The output lines get the data. The timing diagram for the read operations is given in Fig 2.2.
- **WRITE:** Selected fields of the Cache can be written into.The address input to the decoder selects a particular row. The write line is set and the field enable selects a particular field. The input lines have the data to be written. The timing diagram for write operations is shown in Fig 2.3.
- **RESET-VALID:** This operation is used to internally set the Valid bit to the result of the comparison:

```
if (dst < WA|MRV) /*WA includes the extra precision bit */
   then Valid = 0;
```

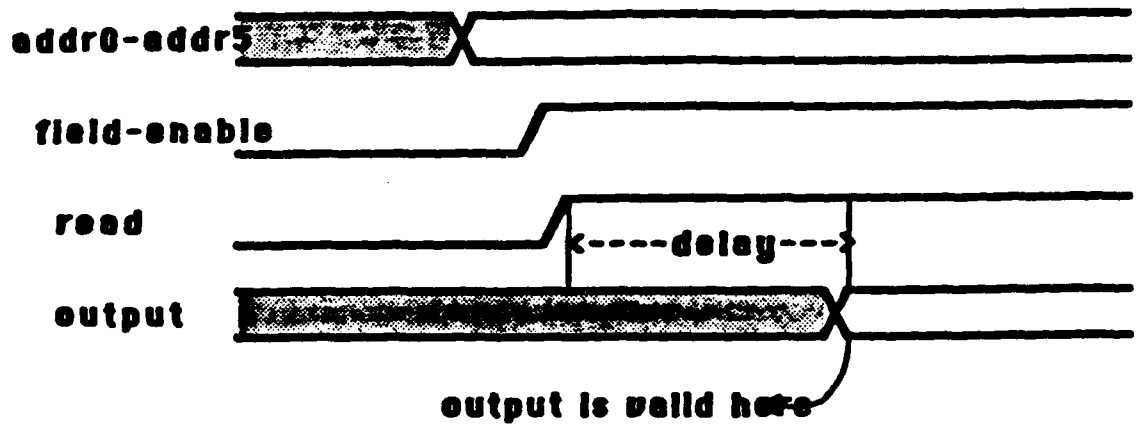
  Here "dst" stands for the value of the data on the WA and MRV input pins (see Fig 2.4). The timing diagram for this operation is given in Fig 2.5.

- **SET-ARCHIVE:** This operation is used to clear the WA and MRV fields when a line is archived.

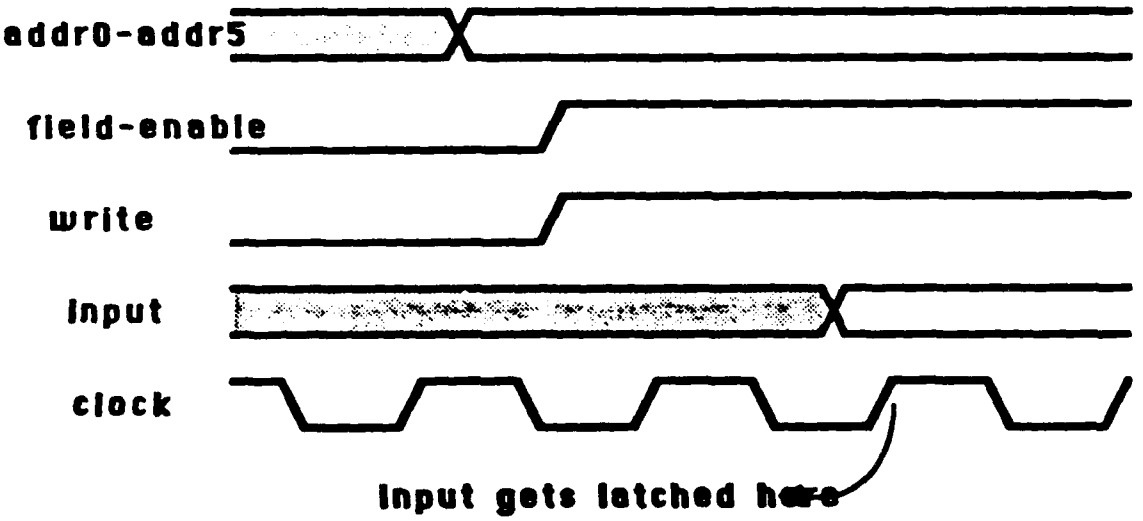
```
if (owa == WA) /* WA does NOT include extra precision bit */
   then  WA = 0; MRV = 0; /* reset extra precision bit of WA field */
```

  Here "owa" is the Old Working Area which is on the WA input pins (see Fig 2.6). The timing diagram for this operation is given in Fig 2.7.

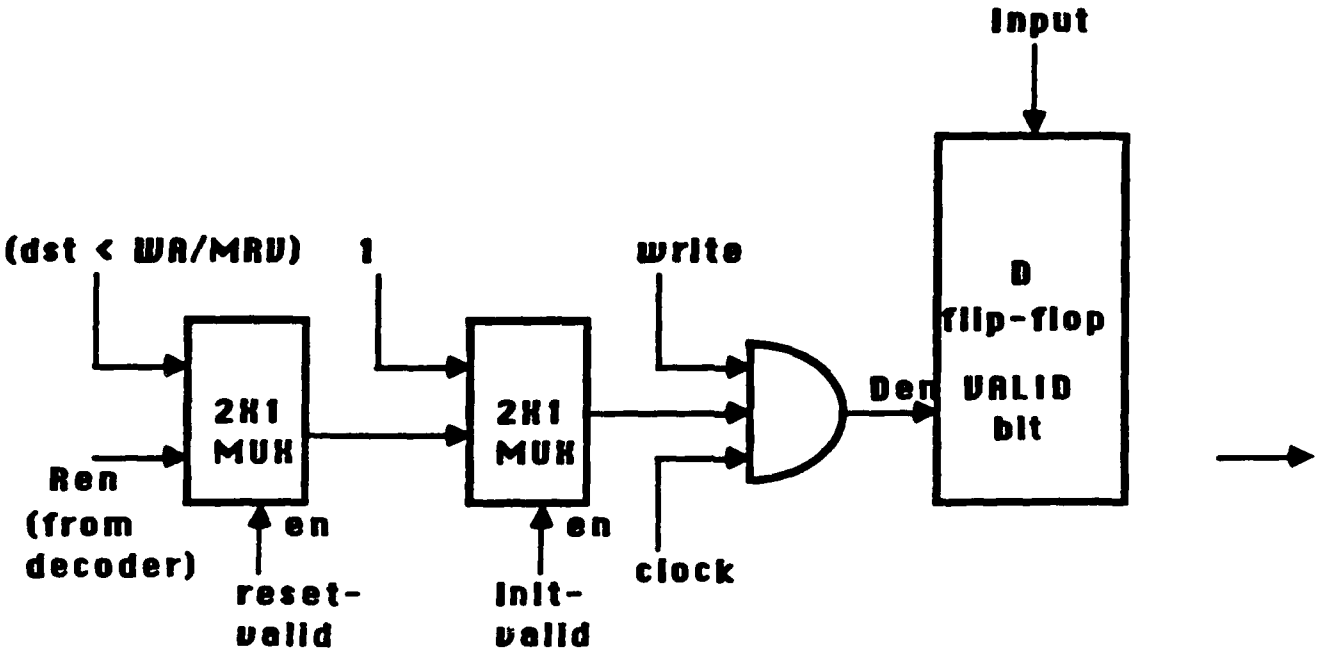
- **WRITE PRECISION BIT:** The write is done on the entire column of WA Precision bits. This is exactly like an ordinary write except that the address lines do not matter.
- **INIT-VALID:** This operation is performed only once at the start when all the rows in the Cache are invalid. The entire column of Valid bits is set to 0.



2.2: TIMING DIAGRAM FOR READ



2.3: TIMING DIAGRAM FOR WRITE



if (dst < WA/MRU) then Valid = 0

2.4 : VALID BIT LOGIC

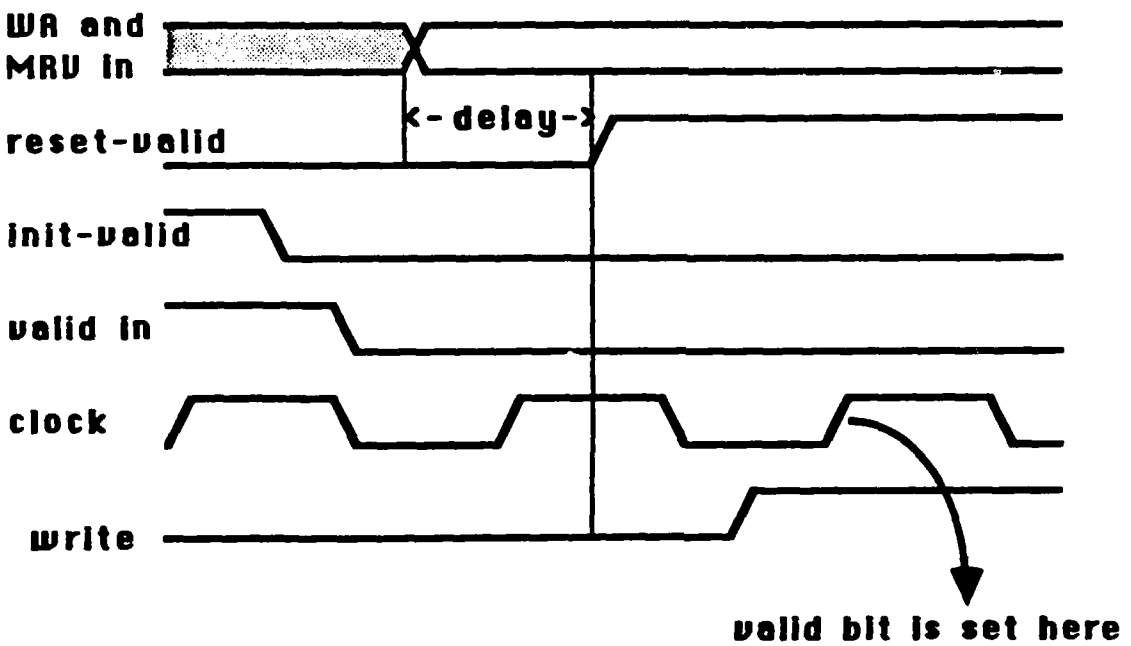
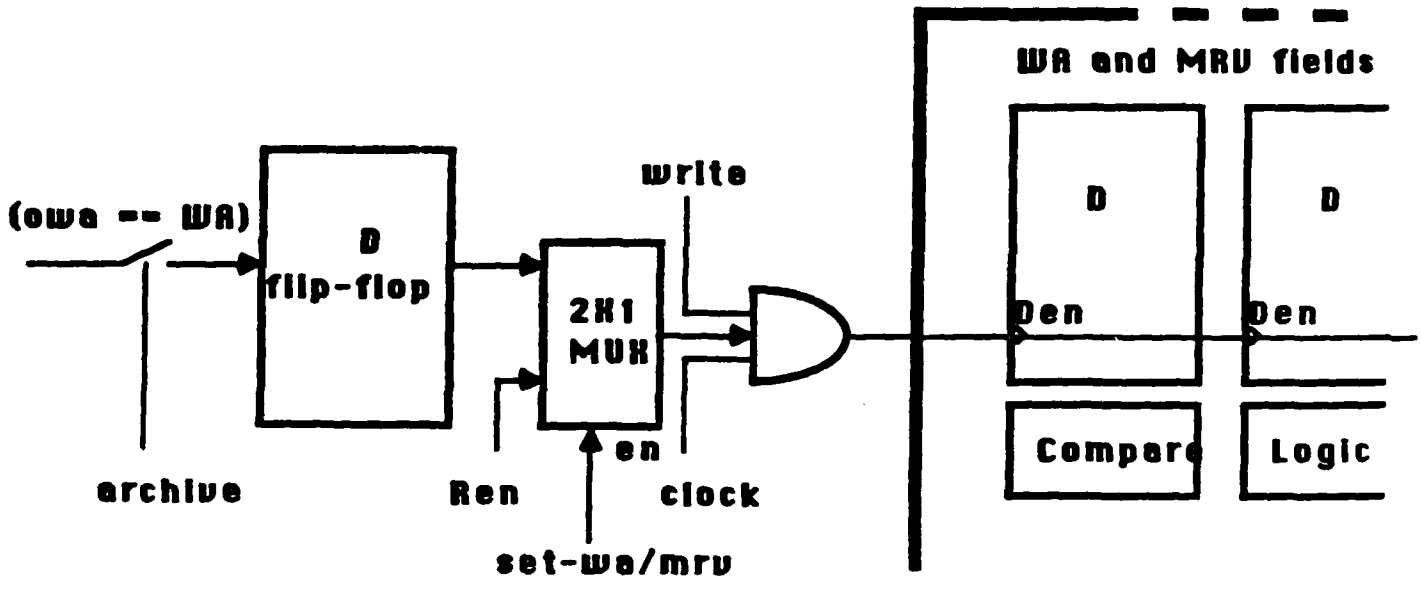
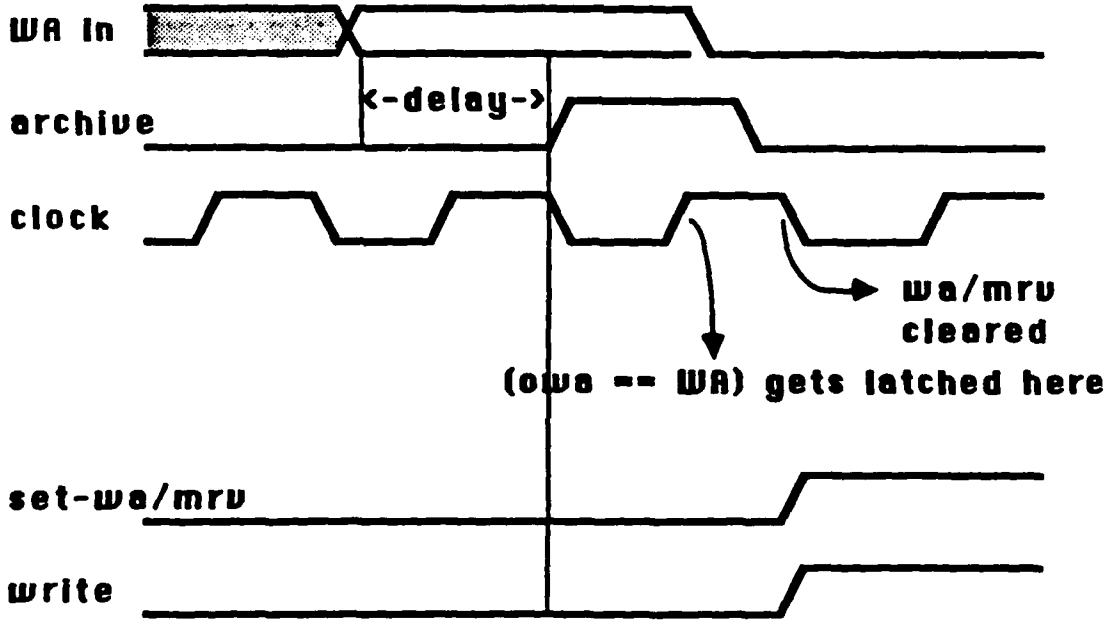


Fig 2.5: TIMING DIAGRAM FOR SETTING VALID BIT



If (owa == WA) then WA=0; MRU=0;

2.6: LOGIC FOR ARCHIVE OPERATION



2.7: TIMING DIAGRAM FOR ARCHIVE OPERATION

Chapter 3

Implementation

The Cache circuit has been implemented in 2.0 micron CMOS technology using **PPL(path programmable logic)** design tools. **Logic simulation** (see Appendix) on the complete circuit has been performed.

The circuit uses a single clock. Great care was applied to minimize the wiring costs. The inverted clock signal runs down vertically on one side of the circuit, and medium sized buffers are placed on this main bar wherever a horizontal branch is required.

Each bit in a row of the Cache was implemented as a D flip-flop with tristated outputs. One of the most important considerations while designing the circuit was wiring. The input and output lines have to run through all the rows of the Cache. We opted for the simplest arrangement where corresponding fields of the rows were directly below one another. This simplified the layout and also shortened the wires. In doing so, we used up some extra silicon. Since this is a prototype chip, ease of testing and simplicity were given priority.

3.1 Simulation and Testing

Using the available tools for **logic simulation**, the complete circuit was checked for logical errors. Functional blocks were simulated individually after they had been designed. Integrating these blocks into the final circuit required testing them again in the new environment to detect errors in wiring. Finally, an exhaustive simulation was performed on the final circuit. This is

documented in the appendix.

3.2 Statistics

The specifications of the Cache chip are as follows:

- Technology CMOS, 2.0 micron
- Number of pins 27
- Number of PPL columns 157
- Number of PPL rows 160
- Number of transistors 10,000
- Minimum operating frequency 4.0 Mhz

Chapter 4

Conclusion

The goal of this project was to implement the Cache in VLSI. This is intended to be a prototype chip. The environment in which it will operate was constantly evolving at the time of writing of this report. In fact, the specifications of the Cache itself changed as we were implementing it. A strong case was made in the introduction for the implementation of rollback capabilities in hardware. As of today, no hardware exists to support rollback capabilites in distributed simulation environments. Further research on this topic is underway at the Computer Science department of the University of Utah.

Bibliography

[1] Richard M.Fujimoto, Ganesh C.Gopalakrishnan and Jya-Jang Tsai. The Roll Back Chip: Hardware Support for Distributed Simulation using Time Warp. *Department of Computer Science, University of Utah*, October 1987.